# TCSS 422 Computer Operating Systems
# Programming Projects Overview

Design and implement a program that simulates some of the job scheduling, CPU scheduling, and deadlock avoidance of an operating system. This document describes the **final** program. Following this overview are the details for each of the three specific projects that must be turned in. Each of those three projects is a portion of this overview description. Note that the first two projects are your own code, written from scratch. The third project is a modification of the code of another student in class done to add a feature to that program.

**For each project, an input data file will be made available to you to be run with your program.** Note that you must allow the user (me!) to specify an input file name, including a path, and an output file name, including a possibly different path. You must accept this information as input to your program; I should not be required to change your code and re-compile it to get this feature to work!

**For each project, you must turn in the following:**

1) A copy of a printout of your source code (with your name!). For programming projects 2 and 3, the code adding additional functionality must be clearly marked off.
2) A copy of the program output on each data file when your program has been run with the posted input.
3) A copy of the printout of your "readme" file containing instructions for running your program **from the command line**, instructions on where the input and output file names are entered (the command line? at a prompt? in a window?), instructions on where the input and output file paths are entered (as part of the file name? separately?), whether you expect that your program runs correctly, the actual time (in hours) it took you to design, code, and debug the project, and any other pertinent information.
4) A CD or floppy disk labeled with your name containing:

- Source code (.java files)
- Executable files (.class files)
- Your "readme" file.

**PLEASE** attach your floppy disk to your printouts with a clip or a folder.

**For each project, grading will be as follows:**

| | |
|---|---|
| Program runs correctly (without error) | 30% |
| Source code layout and comments | 25% |
| Overall design and efficacy | 45% |
| TOTAL | 100% |

**Besides the general categories listed above, points will be taken off your grade for the following:**

1) Late programs (see syllabus for details).
2) Not all pieces of the program listed above are turned in (for example, if you turn in your source code but not your executables or if you do not include in your "readme" whether your expect your program to run correctly or the time it took you to write your program).
3) The output you turn in does not match the output I get when I run your program.
4) For programming projects 2 and 3, new code that adds additional functionality is not clearly marked.

## Project description
The input stream to the program describes a set of arriving jobs and their actions.

When a job arrives, one of three things might happen:
1. If there is not enough *total* main memory or *total* number of devices in the system for the job, the job is rejected.
2. If there is not enough *available* main memory for the job, the job is put in one of the Hold Queues, based on its priority, to wait for enough available main memory.
3. If there is enough main memory for the job, then a process is created for the job, the required main memory are allocated to the process, and the process is put in the Ready Queue.

Preallocation of memory is required, but device allocation is dynamic. Banker's algorithm is used for device allocation.

When a job terminates, the job releases any main memory. The release of main memory may cause one or more jobs to leave one of the Hold Queues and move to the Ready Queue.

When a job terminates, the job releases any devices held. The release of devices may cause one or more jobs to leave the Wait Queue and move to the Ready Queue.

The two Hold Queues are based on priority. There are two external priorities: 1 and 2, with 1 being the highest priority. Priority is only used for the job scheduler, not the process scheduler.

Job scheduling for Hold Queue 1 is Shortest Job First (SJF).
Job scheduling for Hold Queue 2 is First In, First Out (FIFO).

Process scheduling will be Limited Round Robin. Once a job has exceeded the threshold of CPU time, it is considered a long job and may not run until the Ready Queue is empty. At that time, the Long Queue becomes the Ready Queue and Round Robin (with a quantum) process scheduling is used. The Long Queue is FIFO.

The Hold Queues are checked FIRST (before the Long Queue) upon a job completion.

Jobs may acquire and release devices during executions. A device can only be used by one process at a time, because it is a serial (not a random access) device (tapes, printers, etc.)

When a running job requests devices, the Banker's algorithm is used to determine if the request can be satisfied. If the request cannot be satisfied, the process is immediately switched from the CPU to the Wait Queue. If the request can be satisfied, the process is immediately switched from the CPU to the Ready Queue or the Long Queue.

Whenever a running job releases devices and the Wait Queue is not empty, the Banker's algorithm is executed in FIFO order on each job in the Wait Queue to determine if any jobs in the Wait Queue can be allocated its last request of devices. If necessary, the entire Wait Queue is checked to restart as many jobs as possible. A similar remark applies when a job that holds devices terminates, because a job implicitly releases devices upon termination.

The Wait Queue is checked FIRST (before the Hold Queues) upon a job completion.

**Input specification**
The input to your program will be text. Each line in the file will contain one of the commands listed below. Each command consists of a letter in column one followed by a set of parameters. Each text file contains one or more type "C" (system configuration) commands. There will always be *exactly* one blank after each number in the input file.

All input will be *syntactically and semantically* correct, but you should detect and report the following errors, which may occur in the input data:

- The time on a command is either less than or equal to the time on an earlier command.
- A job requests or releases a device when it is not running.

**1. System Configuration:**

C 9 M=45 L=6 S=12 Q=1
This example states that the system to be simulated starts at time 9, and that the system has a main memory consisting of 45; a time excess (threshold) of 6 determines long jobs; the system has 12 serial devices; and the system has a time quantum or time slice of 1.

**2. A Job Arrival:**

A 10 J=1 M=5 S=4 R=5 P=1
This example states that job number 1 arrives at time 10, requires 5 units of main memory, holds no more than 4 devices at any point during execution, and runs for 5. The priority of the job is 1.

**3. A request for devices:**

Q 102 J=3 S=4
This example states that at time 102, job number 3 requests 4 devices. A job *only* requests devices when it is running on the CPU.

**4. A release for devices:**

L 106 J=5 S=1
This example states that at time 106, job number 5 releases 1 device. A job *only* releases devices when it is running on the CPU.

**5. A Display of the current system status in *readable* format (with headings and properly aligned):**

D 11
This example states that at time 11 an external event is generated and the following should be printed:

1. The current time, current available memory, and current available devices (for all displays EXCEPT "D 999999") OR "Final system state", current available memory, and current available devices (for a "D 999999" command).
2. A list of completed jobs, **ordered by job ID**. For each completed job, print a table with the job ID, arrival time, finish time, and turnaround time.

Display items 3-8 for all display commands EXCEPT "D 999999."

3. A list of all job IDs on Hold Queue 1 in the order they are on the queue.
4. A list of all job IDs on Hold Queue 2 in the order they are on the queue.
5. A list of jobs on the Ready Queue in the order they are on the queue. For each job on the queue, print a table with the job ID, run time (total time needed to finish the job), and time accrued (number of time slices used so far on the CPU).
6. The process running on the CPU, including its job ID, run time (how many slices run so far), and time left (how much more time until the job is completed).
7. A list of jobs on the Wait Queue in the order they are on the queue. For each job on the queue, print a table with the job ID, , number of devices held, and number of devices wanted.
8. A list of jobs on the Long Queue in the order they are on the queue. For each job on the queue, print a table with the job ID, run time, and accrued time.

Display item 9 for ONLY a "D 999999" command.

9. The system turnaround time only at the last display. Assume that the input file has a "D 999999" command at the end, so that you dump the final state of the system. "D 999999" is the very last event of the input file and subsequently of the system. Be sure and finish all processing of all jobs before displaying the final state of the system.

**Summary of Events and their Consequences (transitions):**

**External**

**Initial** – simply start the simulation and initialize the "clock"

**Arrival** – possible actions:

1. Job is rejected if the max memory is greater than total memory OR if the max devices are greater than system devices.
2. Job moves to Hold Queue if there is not enough memory available or if there are jobs on the Hold Queues.
3. Job moves to Ready Queue if there is available memory

**Request** – "pretend" to make the request, call Banker's and Banker's will either allow the request to be granted (job goes to the Ready Queue or Long Queue) or Banker's will determine an unsafe state (job goes to the Wait Queue). If the job moves to the Wait Queue, there is a possibility that the Long Queue could become the Ready Queue (i.e. the job going to wait was the last job in the Ready Queue and the Long Queue is not empty).

**Release** – job releases devices, the wait queue is then checked (in FIFO order) to see if one or more jobs can move to the Ready Queue. If a job moves from the Wait Queue, there is a possibility that the Long Queue will be affected.

**Display** – the contents of the queues, etc., will be dumped. The display does NOT cause the running job to be interrupted. It lets it finish its time quantum.

**Internal**

**End of quantum** – a context switch occurs.

**Completion** – the job releases its devices (this can cause one or more jobs to move from Wait to Ready, or Wait to Long). The job releases its memory (this can cause one or more jobs to move from Hold to Ready).

Whenever a job moves to the Ready Queue (arrival, completion, release), the Long Queue may be affected.

**Additional specs: All Programming Projects**

There are two types of events in this program: internal events (preemptions and completions) and external events (all events that come from the input file). You should have a loop in your program that calculates the time of the next event (internal or external). If there is BOTH and internal AND an external event at the same time, then process the internal event BEFORE the external event. Notice that is this is not strictly followed, your results will not match the expected output to grade your project!

The end of a time slice is an internal event. You may assume a context switch will take zero time.

There will never be two external events at the same time. If a second external event arrives with the same time or an earlier time as a previous one, ignore the second event.

Absolutely under no circumstances do you want to read the entire input file in the beginning of the program (i.e. pre-process the input file).

Use linked lists for your queues rather than arrays, vectors, or circular arrays; you may get the linked list code from another source; you do not need to write that code yourself.

You will be graded in part on the maintainability of your code. Do not embed number constants in the code.

Information hiding is important.

Single-letter or non-descriptive variable names in your code will affect your grade. Use descriptive names.

If more resources are needed than the system contains (not available; actually CONTAINS), then do not even consider the jobs. (Kick it out and do not "count" the job).

**Additional specs: Programming Project 2**
Implement the Hold Queues as a sorted linked list.

If jobs have same run-time and same priority, use FIFO scheduling in Hold Queue 1 (FIFO within SJF).

When a job completes, it releases main memory. Check the two Hold Queues before the Long Queue.

When memory is released by a job completing, dequeue job(s) from the FRONT of Hold Queue 1. If Hold Queue 1 is empty, dequeue job(s) from the FRONT of Hold Queue 2. ONLY check Hold Queue 2 when Hold Queue 1 is empty. Check to see if jobs can be taken off the Hold queues when a job completes OR when a job arrives.

If either Hold Queue is not empty, place arriving jobs in a Hold Queue. Do not take an arriving job and put it on the Ready Queue in front of jobs that should have a higher priority (jobs already in the Hold Queues).

Moving jobs from the Hold Queues or Long Queue onto the CPU or the Ready Queue are INTERNAL events. Therefore, in your programs, when you need to add jobs to the CPU or Ready Queue you should look for jobs in the following order: (1) Hold Queue 1, (2) Hold Queue 2 (if Hold Queue 1 is empty), (3) Long Queue (if Ready Queue is still empty after checking the Hold Queues), (4) external jobs coming into the system (if you have enough available memory to accommodate them). Note that this means that you would take a job off Hold Queue 2 BEFORE an external job of priority 1. It also means that you might execute a job on the Long Queue for one quantum BEFORE an external job gets put on the Ready Queue.

When a job preempts off the CPU, then check whether to move that process to the Long Queue or not. Do not preempt a job off the CPU simply because it has executed enough times that it is now a long job. Allow jobs to finish their time quantum before preempting them and putting them in the Ready Queue or the Long Queue.

When a process is in the Long Queue, it does NOT give up it resources.

When you are checking whether to put a job back on the Ready Queue or on the Long Queue, check for an accumulated value greater than OR EQUAL TO the threshold value.

Do not actually move jobs from the Long Queue to the Ready Queue. Instead, when you need a job to execute on the CPU, if the Ready Queue is empty, simply put a job on the CPU from the Long Queue.

**Additional specs: Programming Project 3**
Devices are only requested by jobs while running on the CPU, and if this happens, the job’s time slice is interrupted. A job that requests a device in the middle of a time quantum, whether it gets the device or not, does NOT finish its time quantum. It blocks immediately.

Devices are only released by jobs while running on the CPU. In this case, the job’s time slice is NOT interrupted.

On an arrival, the “S=” denotes the maximum number of device(s) the job will ever use. Use this number in your Banker’s table.

Check the Wait Queue before the two Hold Queues.

The Wait Queue is FIFO.

(Modified from Dr. Sallie Henry at Virginia Tech)

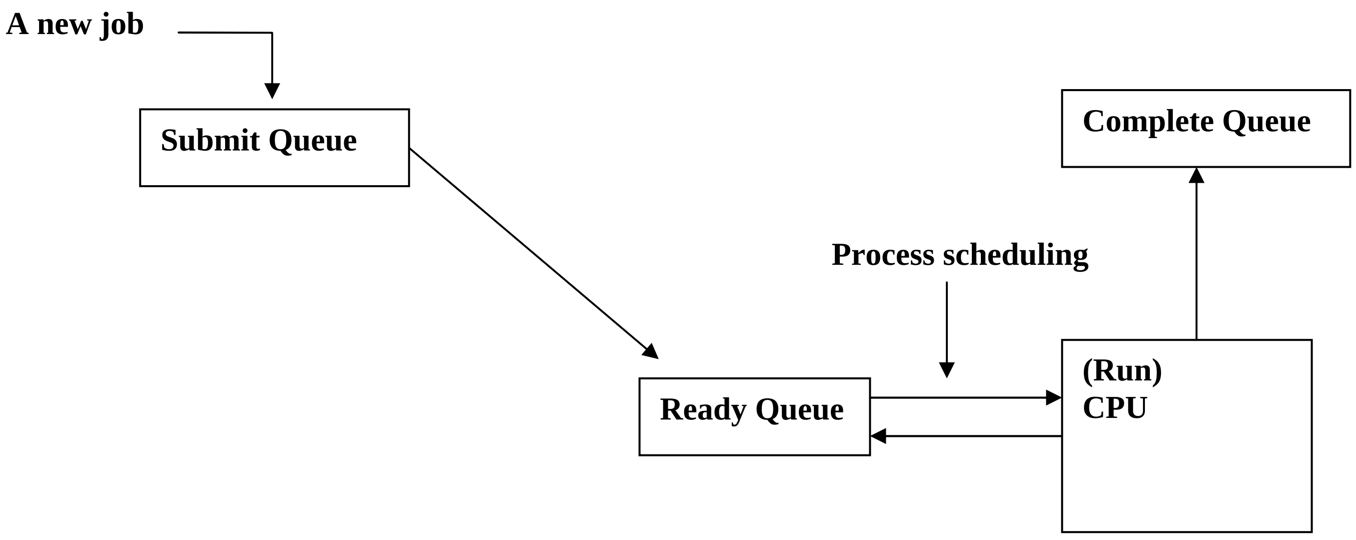
**Programming Assignment #1**
**Process Scheduling**

Programming project #1 consists of process scheduling. This program will be expanded to create projects #2 and #3. Stubs may be used in this project to prepare for future development.

Process scheduling is Round Robin.

For this assignment, input shall be handled as follows:
1. System configuration:
Set the clock. Store the configuration information.
2. A job arrival:
Set the clock if the time on the job is greater than the time on an earlier command. If the amount of main memory required is less than the main memory in the system and the maximum number of devices needed is less than the number of devices in the system, put the job in the Ready Queue. Otherwise, reject the job.
3. A request for devices:
Ignore the request.
4. A release for devices:
Ignore the request.
5. A display of the current system status in readable format:
Display all the fields, with blanks in the fields not applicable for this assignment.

## Graphic view of Assignment #1



**Programming Assignment #2**
**Job Scheduling**

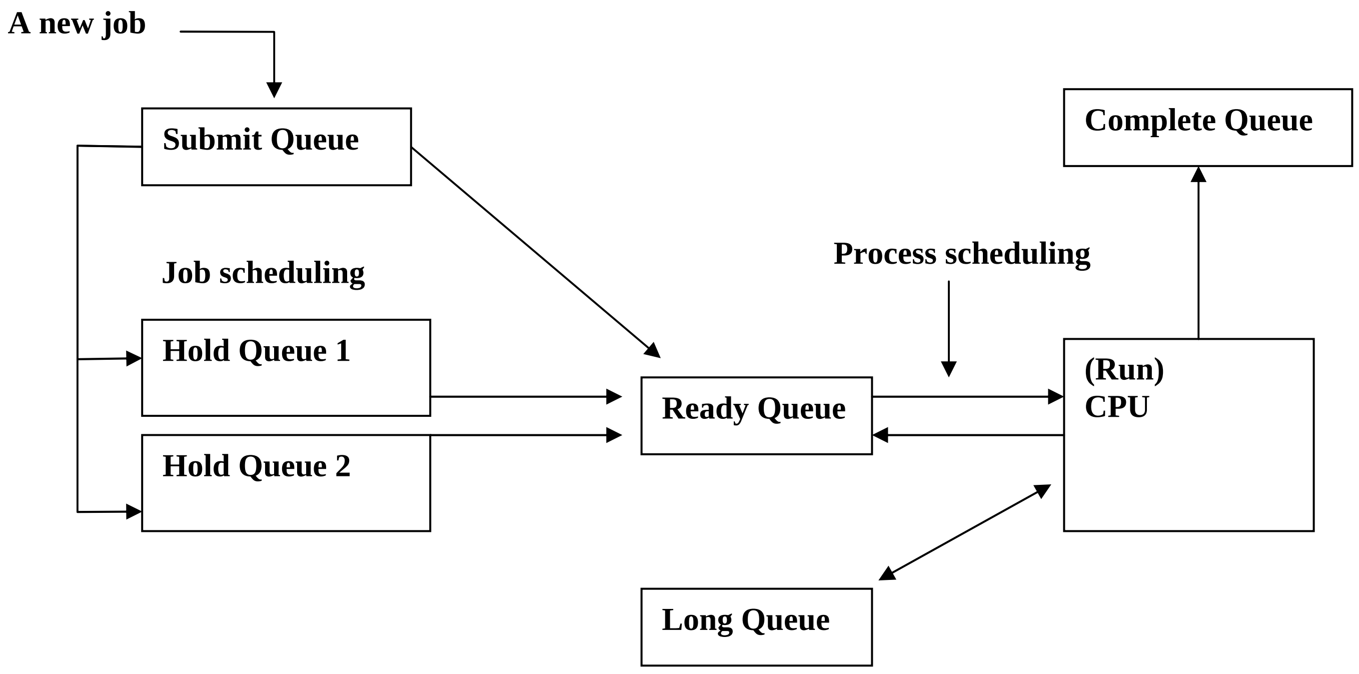
Add job scheduling to programming assignment #1. Add the Hold Queues and the Long Queue.

Process scheduling is Limited Round Robin.

Job arrival shall be handled as follows:
1) Reject the job if there is not enough *total* main memory or *total* number of devices in the system for the job.
2) If there is not enough *available* main memory, the job is put into one of the Hold Queues (based in its priority) to wait for enough available main memory.
3) If there is enough main memory for the job, then a process is created for the job, the required main memory and devices are allocated to the process, and the process is put in the Ready Queue.

System configuration and display of the current system status shall be handled as the input specification in the overview document. Input that requests or releases devices shall be ignored.

## Graphic view of Assignment #2

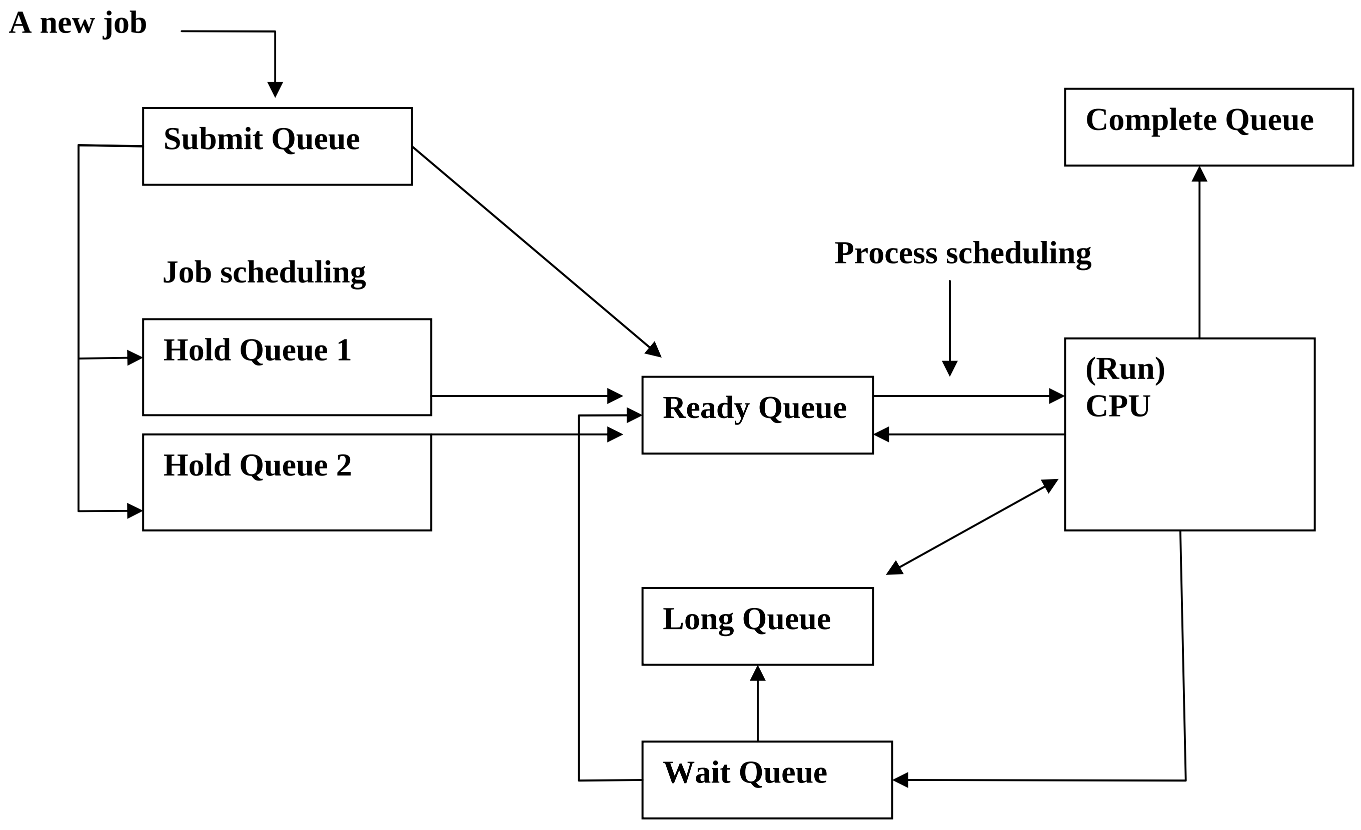


## Programming Assignment #3
## Deadlock Prevention

This assignment shall conform in full to the overview document.

This assignment will give you an opportunity to practice following and modifying code that is not your own, which is an invaluable skill in the "real world." Disks/CDs will be collected from all students for assignment #2, and will be re-distributed to another student. You will add the functionality for assignment #3 to a program **that you did not write from scratch**. You should first run the program given to you and follow the functionality through the code. Any bugs you find should be fixed. You should then add the functionality of deadlock prevention to the code given to you **without** re-writing large portions of the given code.

### Graphic view of Assignment #3


A new job
Submit Queue
Complete Queue
Job scheduling
Process scheduling
Hold Queue 1
Ready Queue
(Run)
CPU
Hold Queue 2
Long Queue
Wait Queue